COMSW4995 Applied Machine Learning

Final Report

Group 26

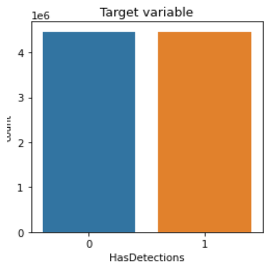
Nicklaus Ong Jing Xue (no2367)

Siyu Li (sl4826)

Srividya Inampudi (si2396)

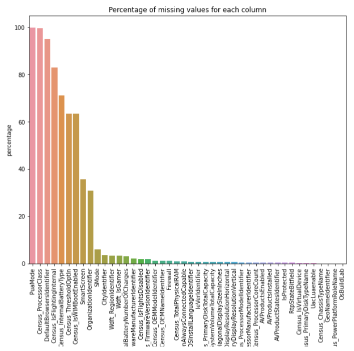
Xianmeng Wang (xw2746)

Preprocessing and Feature Engineering

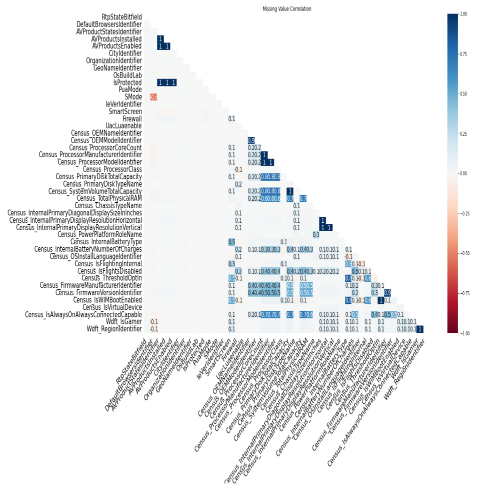


Data Type: The training set has about 9 million tuples, 53 numerical columns, 30 categorical columns. The target variable is 'HasDetections'. This is a binary classification problem.

Imbalanced columns: The target variable is balanced. As the right plot shows the percentage of two labels in the target variable are equal. Thus we do not need to resample. **Drop highly imbalanced categorical columns** whose majority category is over 99%.



Missing values: The left bar plot shows the percentage of missing values for each column. **Drop features having over 95% missing values.** The right heatmap shows correlations of missing values between pairs of columns. We observe some columns' missing values have high positive correlations, for example, the correlation missing value between 'AvgProductInstalled' and 'AvgProductEnabled' is 1, which shows the values in the two columns are always missing together.



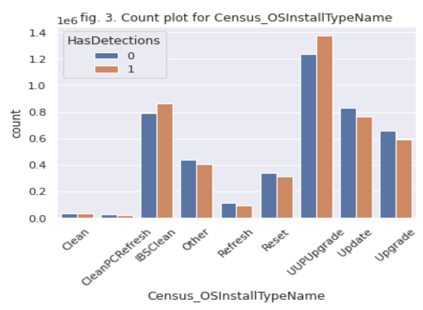
Creating an is_missing for every column would introduce multicollinearity. We choose the replacement. For numerical features: **Replace missing values as a special number outside of range** (minimum value minus 1 which decision trees can handle). For categorical features: **Replace missing values as 'unknown'**.

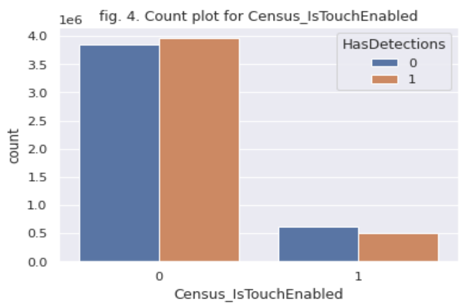
Data preprocessing: For Categorical features: Change values to lowercase version uniformly; Merge different spelled but same meaning values to avoid reduplication; Set categories very rarely occur in each feature as one 'other' category to simplify; Label features with high cardinality as numerical values representing its frequency to avoid curse of dimensionality; One-hot encoding other features. For numerical features: Standardize numerical features to prepare for training machine learning models.

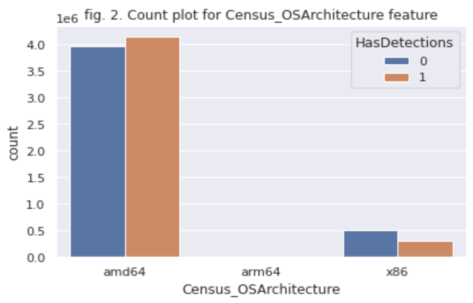
Exploratory Data Analysis

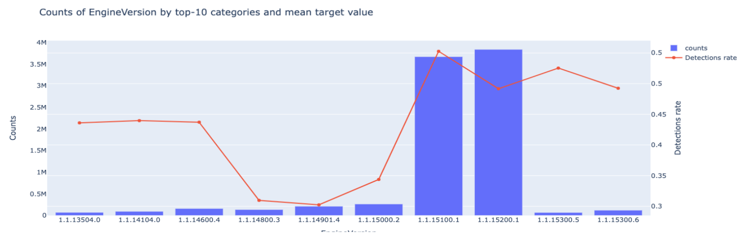
Count plots help us understand the data representation of various categories of a particular feature. In fig. 2. We can see that Most of the machine architecture is amd64 and most of the amd64 architecture machines have detected the malware. Processors with amd64 architecture can be more vulnerable from malware as the detection rate of malware is high as compared to x86 and arm64 architecture. In fig. 3. We can see that Most of the machines are with install type UUPUpgrade, Update, Upgrade and IBSClean. Rate of infection is lower for machines with install type Update and Upgrade compared to

UUPUpgrade and IBSClean. In fig. 4. We can see that Microsoft has many more computers that are not touch devices. The rate of infections is lower for touch devices.

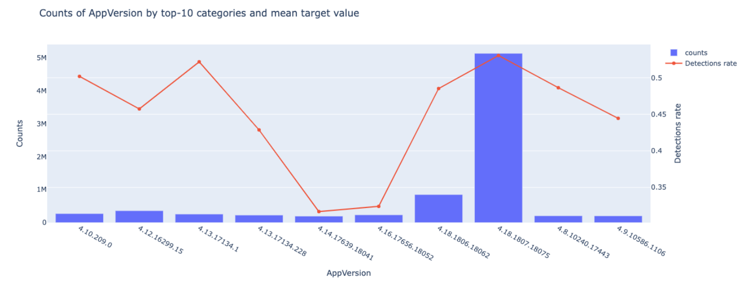








Detection rates for different EngineVersion - We can see that two categories take 84% of all values. The difference in detection rates is noticeable. Other categories have different detection rates, but it is simply due to the low number of samples in them.



Detection rates for different AppVersion - In this case we have one main category with 0.53 detection rate, other categories are much smaller. We can see that some versions have higher detection rate even though their count is less it means those versions are most susceptible to malware.

Feature Selection, Model Evaluation and Machine Learning

Variable Selection: We drop features where the correlation between variables is more than 0.90 (removed 5 variables in total). Also, according to the principle of parsimony, we would prefer simpler models. To test if simple models can achieve better results, we use sklearn SelectKBest to select the top 20 variables using Chi-Squared test to see if a parsimonious model with fewer variables performs well. We will compare using top 20 variables against using all variables.
Train Test Split: We used random sample without replacement, train test split of 80% train and 20% test, and min max scaling (fit transform on the train set, and transform only on the test set).
Model Evaluation: For model evaluation, we used AUROC, Precision, Recall and Accuracy, which provides a comprehensive evaluation of our model performance. For hyperparameter optimization, we used AUROC.
Unique Consideration: Our chosen dataset presents an interesting problem; with the large size of our dataset, training time and processing time is significantly longer. Because of this, we have to consider efficient models, and have to be selective with hyperparameter optimization.
Baseline Models With Basic Preprocessing: (Include all columns, fill na with median for numerical, fillna with majority class for categorical, target encoding for all categorical variables). Train Logistic Regression and Random Forest with 20 trees. SVMs could not be implemented due to extremely long training time.
Advanced Model with Advanced Preprocessing: As described in preprocessing (handling missing values, dropping

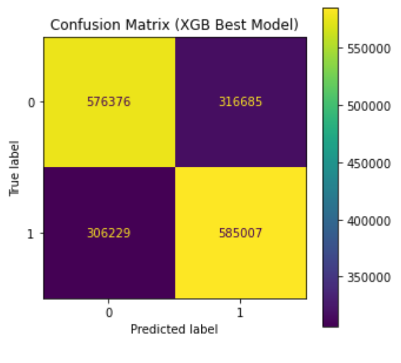
correlated features, handling majority class etc.). A point to note is that we found that target encoding works better than our original method of one hot encoding/frequency encoding, so **we used target encoding**. Trained Xgboost: a state-of-art tree to handle tabular data with categorical variables, and LightGBM: fast, handle sparsity, handle categorical variables.

Model	Accuracy	Recall	Precision	AUC
Logistic Regression (Baseline)	0.632	0.645	0.637	0.686
Logistic Regression (Baseline with top 20 features)	0.629	0.646	0.625	0.684
Random Forest (Baseline)	0.632	0.584	0.646	0.687
Random Forest (Baseline with top 20 features)	0.632	0.583	0.636	0.677
LightGBM (With Advanced Preprocessing)	0.646	0.653	0.644	0.703
LightGBM (With Advanced Preprocessing and optimization)	0.648	0.655	0.645	0.705
XGBoost (With Advanced Preprocessing)	0.651	0.656	0.649	0.709

We could only perform hyperparameter optimization with LightGBM since the training time of the ML technique was fast enough. In addition, we subsampled 100,000 data points for the optimization, and Bayesian optimization was performed using CV of 2 (chose 2 due to time constraints and the fact that splitting the data into half still gives an adequate number of data) and optimizing on auc.

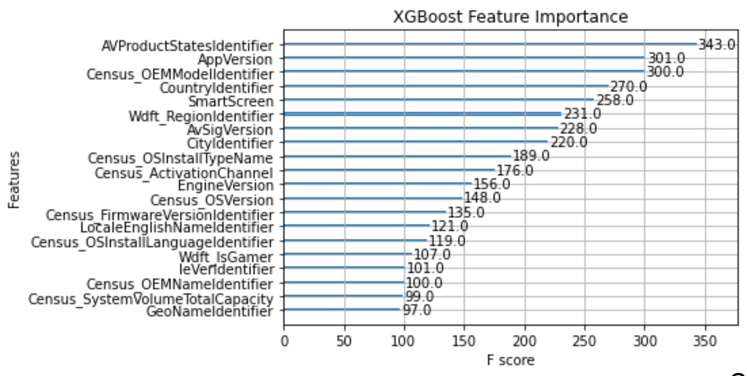
1. For the baseline models, we used minimal preprocessing. We also compare a case of using all variables versus using just the top 20 features. As we can see, using the top 20 features versus using all features resulted in worse performance for both logistic regression and random forest (lower AUC, precision, while accuracy and recall are the same). Using all features would have more meaningful results.
2. Our advanced preprocessing and advanced models outperformed the baseline models by a large margin of 0.709 AUC versus 0.68 AUC.
3. Bayesian Optimization on LightGBM improved model performance!
4. Our chosen final model is XGBoost with Advanced Preprocessing since it outperformed all other models on every evaluation metric!

We submit the predictions of the model with best AUC (XGBoost) to kaggle leaderboard. The public score is 0.670 while the private score is 0.620, which is lower than the validation result. We feel that the test data is harder to predict. Following the prediction, we would like to show some plots about information about the prediction result.



On the left, we plot the confusion matrix for XGBoost with advanced preprocessing, our best model. As can be seen, our model does not have significant bias towards predicting any single class. This is a good result as it shows our model can identify normal computers without malware, and computers with malware.

We are also interested in analyzing the feature importance to assess if our model is sensible. Below, we plot the feature importance for XGBoost. As we can see on the right, AVProductStatesIdentifier, and AppVersion are ranked as the best two features. This makes intuitive sense as AVProductStatesIdentifier refers to the specific configuration of a user's antivirus software; some configurations might not be as good at protecting against malware, and AppVersion is also important, which might show that certain versions have vulnerabilities. This is highly intuitive, and through this research we have also found insights we can apply to protect computers from malware; we should make sure we have a good antivirus software configuration and have an updated app version that has low vulnerabilities.


XGBoost Feature Importance
Features
F score
AVProductStatesIdentifier 343.0
AppVersion 301.0
Census_OEMModelIdentifier 300.0
CountryIdentifier 270.0
SmartScreen 258.0
Wdft_RegionIdentifier 231.0
AVSigVersion 228.0
CityIdentifier 220.0
Census_OSInstallTypeName 189.0
Census_ActivationChannel 176.0
EngineVersion 156.0
Census_OSVersion 148.0
Census_FirmwareVersionIdentifier 135.0
LocaleEnglishNameIdentifier 121.0
Census_OSInstallLanguageIdentifier 119.0
Wdft_IsGamer 107.0
IeVerIdentifier 101.0
Census_OEMNameIdentifier 100.0
Census_SystemVolumeTotalCapacity 99.0
GeoNameIdentifier 97.0